CSE474 Embedded Systems

Lab 4

Student ID	Name	Email
1471626	Daniel Sullivan	dtsull16@uw.edu
1826649	Jeffrey Lee	cn1207@uw.edu

1 Problem

For our Lab 5, the problem we wanted to address was implementing a control system for a smart parking garage. Smart parking garages increase convenience and decrease staffing costs for parking. Most simply, they automatically detect vehicles and open the gate. At a more complex level, they can keep track of which vehicles are coming in to the garage, how often they are parking, and determine which vehicles are allowed to park in the garage. For the purposes of this lab, we decided to focus on the embedded systems portion of this control system, namely detecting vehicles and opening the garage gate.

2 Procedure

We decided to implement this using a HX-3CC servo motor and OV7670 camera [4] on our TM4C Microcontroller [2]. We planned on capturing an image with the camera and processing the image with a machine learning library to detect if there is a vehicle in the image, then using the TM4C Pulse Width Modulation (PWM) module to control the motor to open the parking garage gate. Xnor.ai provides numerous machine learning libraries for embedded systems applications with pre-trained machine learning models to detect vehicles in images, which suited our purposes perfectly [5]. This could then be expanded upon with other machine learning libraries to detect things like licence plate numbers to allow even more specific management of the parking garage. For the purposes of demonstration, however, we 3D printed a model parking garage and attempted to detect the shape of model cars in the garage.

The OV7670 camera uses Serial Camera Control Bus (SCCB) protocol [3] to communicate, which is compatible with the Inter-Integrated Circuit (I2C) protocol on the TM4C board. We then needed to provide the camera with a clock input to drive the camera system clock (XCLK). We used another PWM module to create a clock input for the camera module. However, the camera needs a 10Mhz - 48Mhz clock input [4] and the TM4C board can only modulate the PWM module at the rate of its internal system clock. This means that the TM4C can only produce a clock with PWM with a maximum frequency of one half its system clock frequency. We configured the TM4C Phase Locked Loop to run the system clock at a frequency of 66.67Mhz so we could modulate the PWM every 2 clock cycles and produce an output clock of 16.67Mhz for the camera to use as input.

The Xnor.ai image processing library requires a minimum of 300x300 resolution to be able to process an image and detect if it contains a vehicle [5]. So, we used the TM4C I2C module to interface with the OV7670 camera SCCB protocol and set the camera's registers and configure the camera to capture images at a VGA resolution of 640x480 [4]. We could then connect the camera's vertical synchronization (VSYNC), horizontal reference (HREF), and pixel clock (PCLK) outputs to the TM4C board and use those to detect when to capture data from the camera's 8 data output pins. A VSYNC falling edge indicated the camera is outputting a new image frame, and a HREF rising edge indicates the camera is outputting bytes from a new row of pixels. While VSYNC is outputting low and HREF is outputting high, the camera outputs a new byte of image data over its 8 data pins with every tick of its PCLK output. By setting interrupts for the VSYNC and HREF edges, we could then use an interrupt on the PCLK tick to capture the 8 data pin outputs and store that as a byte of image data.

We could use a TM4C General Purpose Timer (GPTM) to sample the camera every 1 seconds and process the image data with the Xnor.ai library to detect if there was a vehicle in the image. If the machine learning library outputted that there was a car in the image, we could modulate the PWM connected to the servo motor to 80% of its previous period which would raise the gate to the parking garage and start a one-shot GPTM timer. After 5 seconds, the GPTM timer would set the PWM back to its original period which would lower the gate again.

3 Problems Encountered

Unfortunately, we ran into a major issue when putting all these components together just prior to the demo. The OV7670 camera outputs images in YCrCb format which requires 3 bytes of data per pixel. For a 640x480 resolution image, this requires $3\text{bytes} * 640 * 480 = 921,600\text{bytes} = 921.6\text{KB}$ of memory to store the image, and the TM4C only has 256KB of flash memory [2]. This meant we could not use image processing to detect vehicles for our parking garage and had to find another method of detection. We decided to remove the camera and machine learning library, and use a HC-SR04 Ultrasonic Sensor [1] to detect if there is a vehicle in front of the parking garage gate. This would not allow for as specific management of the parking garage gate, since any object in front of the gate would trigger the gate to open, instead of just cars, but it would still allow us some method of detection to trigger the gate to open.

The HC-SR04 Ultrasonic Sensor communicates over TTL protocol. When provided with a 10 uS pulse on its trigger (TRIG) input, the sensor sends a series of ultrasonic pulses to detect items in front of it in a 15 degree arc up to 400cm away [1]. The sensor then outputs a pulse over its ECHO pin with a duration proportional to the distance of the object it detected. We used a PWM Timer to interface with the TTL trigger signal and an Input Edge Time Timer to interface with the the TTL response echo signal [2].

We configured the PWM timer to output a 10 uS pulse every 40 ms. On sending out the pulse, we configured an interrupt to start the Input Edge Time Timer to listen for the response signal. We configured the Input Edge Time Timer to interrupt on both the rising and falling edge of the response signal. When the timer interrupted on the rising edge of the response signal, we recorded the timestamp on the timer. Then when the timer interrupted on the falling edge of the response signal, we recorded the timestamp on the timer again and disabled the timer. This difference between the rising edge timestamp and falling edge timestamp is the duration of our response pulse which we could use to determine the distance of the object the ultrasonic sensor detected.

We calculated the distance to the detected object according to the equation on the HC-SR04 datasheet: $\text{distance} = \text{response time} * \text{pulse velocity (340 m/s)}/2$. This allowed us to calculate when an object was within 2cm of the ultrasonic sensor and trigger the gate to open.

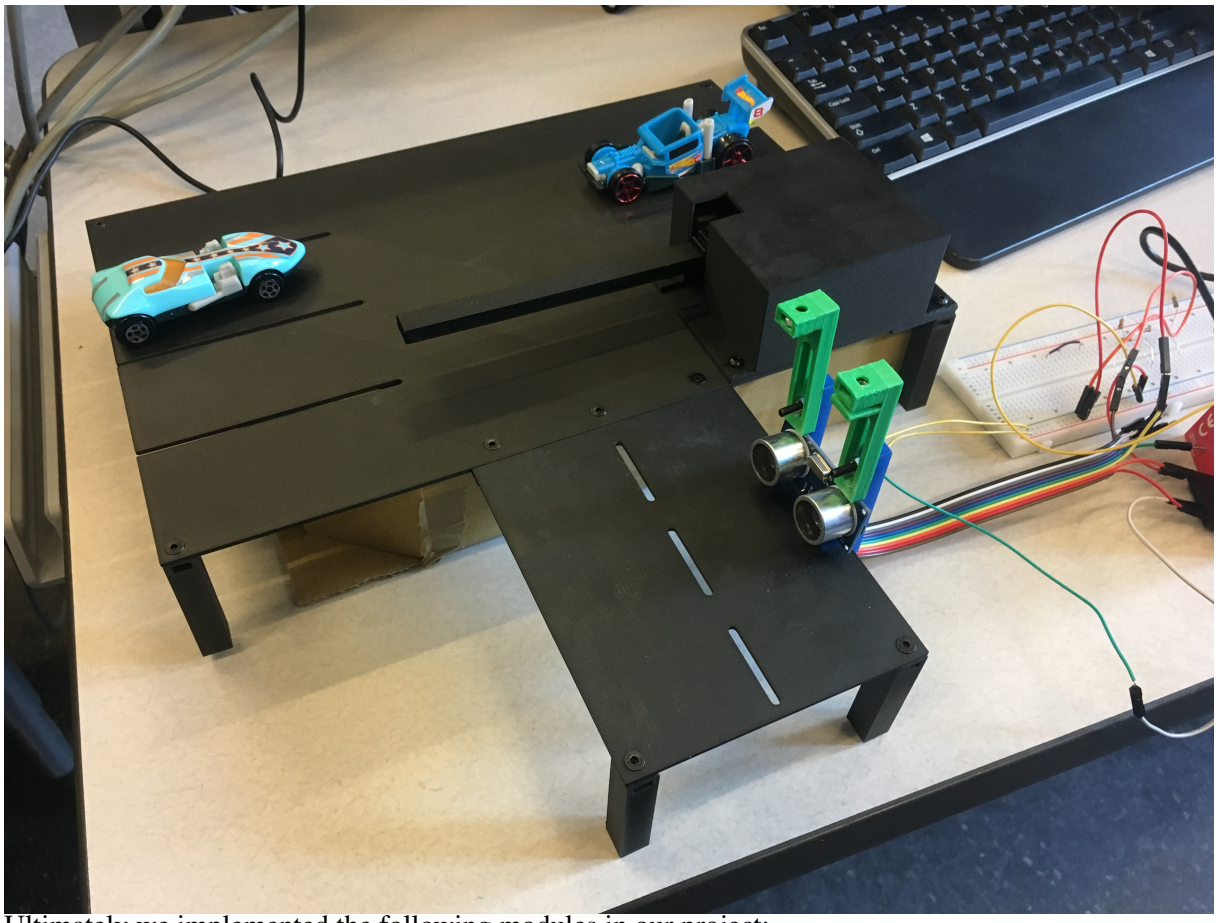
We triggered the gate to open in a very similar fasion to our original design. If the gate was not already open and the the object was within 2cm of the sensor, we modulated the PWM connected to the servo motor to 80% of its previous period which would raise the gate to the parking garage and start a one-shot GPTM timer. After 5 seconds, the GPTM timer would set the PWM back to its original period which would lower the gate again.

Although we did not manage to attain the level of specific control of the smart parking garage, we were still able to implement automated control of the embedded systems component of the smart parking garage system and demonstrate it at a model scale.

4 Results

Although our final system involved a lot of complex interactive components in terms of Timers, PWM modules, and interrupts, it remains fairly simply in terms of its FSM states. The entire system has only two tracked states: gate open or gate closed. The majority of other components involved detecting or implementing the state transitions. The final state diagram is as follows:

Our final model scale demonstration setup is depicted below:



Ultimately we implemented the following modules in our project:

- PWM module
- One Shot Timer
- PWM Timer (for TTL protocol)
- Input Edge Time Timer (for TTL protocol)
- HC-SR04 Ultrasonic Sensor Module

5 Conclusion

We made a lot of modifications to our design (both large and small) throughout the development process and iterated through many different implementations through the process of research and experimentation. Although we were not able to implement our original design, we demonstrated that a variety of approaches can be taken to obtain the same result (to a varying degree of specification). We implemented the an embedded systems control portion of a smart parking garage at a model scale and demonstrated that through this modular engineering approach, various features can be added or swapped to obtain a final product. We demonstrated this by developing our servo motor controller and vehicle detection

components in parallel and then combining them at the end. Even though the vehicle detection component needed to be completely redesigned, the servo motor controller was still able to be used exactly as originally designed.

We could use a similar approach to expand upon our product and add components for web interfacing, databases, or coordination between multiple gates to implement a smart parking garage in a modular fashion to improve ease of use.

6 Appendix

6.1 main.c

```
// AUTHORS: Daniel Sullivan, Jeffrey Lee
// COURSE: CSE 474
// PROJECT: Lab 5
// TITLE: main.c

#include <stdint.h>
// #include <stdio.h>  // For debug info
#include <stdbool.h>
#include "lab5.h"

// Definitions
#define PF2                     0x04    // Trig
#define PF4                     0x10    // Echo

#define PB6                     0x40    // M0PWM0

#define RATE_16MHZ_1HZ          0xF42400  // 16000000
#define RATE_16MHZ_2HZ          0x7A1200  // 8000000
#define RATE_16MHZ_0_5HZ        0x1E84800  // 32000000
#define RATE_16MHZ_0_25HZ       0x3D09000  // 64000000
#define RATE_16MHZ_0_2HZ        0x4C4B400  // 80000000
#define RATE_16MHZ_0_1HZ        0x9896800  // 160000000
#define MAX_RATE_16BIT          0xFFFF  // 16 bit sample rate
#define RATE_16MHZ_10microHZ    160

#define PERIOD                  1200
#define TRIGGER_DIST_CM         10

#define EN0_19                  (1 << (INT_TIMER0A - 16))  // Timer0A
#define EN0_21                  (1 << (INT_TIMER1A - 16))  // Timer1A
#define EN0_23                  (1 << (INT_TIMER2A - 16))  // Timer2A

// Forward declarations
void init_timer0A(uint32_t);
void init_timer1A(uint32_t);
void init_timer2A(uint32_t);
void init_pwm(uint16_t);
void set_period(uint16_t);
void init_portF(void);
```

```
void Port0A_Handler(void);
void Port1A_Handler(void);
void Port2A_Handler(void);

// State
static bool gate_open;

int main() {
  gate_open = false;

  init_portF();  // init PF2 for T1CCP0 and PF4 for T2CCP0
  init_pwm(PERIOD); // init pwm
  init_timer0A(RATE_16MHZ_0_2HZ); // init timer0A (gate timer)
  init_timer2A(MAX_RATE_16BIT); // init timer2A (ultrasonic read timer)
  init_timer1A(MAX_RATE_16BIT); // init timer1A (ultrasonic trigger timer)

  while (1);

  return 0;
}

// close gate again
void Timer0A_Handler() {
  TIMER0_ICR_R |= TIMER_ICR_TATOCINT; // clear interrupt flag
  set_period(PERIOD); // close gate
  gate_open = false;
}

// enable timer2 when triggering sensor
void Timer1A_Handler() {
  TIMER1_ICR_R |= TIMER_ICR_CAECINT; // clear interrupt flag
  TIMER2_CTL_R |= TIMER_CTL_TAEN;  // enable timer2A
}

// Detect duration of high level echo
void Timer2A_Handler() {
  TIMER2_ICR_R |= TIMER_ICR_CAECINT; // clear interrupt flag

  static uint32_t rising_edge_time = 0;
  if (rising_edge_time == 0) {  // rising edge
    rising_edge_time = TIMER2_TAR_R;
  } else {  // falling edge
    int32_t clock_ticks = TIMER2_TAR_R - rising_edge_time; // get time of level
    rising_edge_time = 0;  // reset rising edge
    TIMER2_CTL_R &= ~TIMER_CTL_TAEN;  // disable timer2

    // distance = (high level time * velocity of sound (340M/S) / 2
    double dist_cm = (clock_ticks * 0.0010625);  // convert to distance in cm
    if (dist_cm > 0 && !gate_open && dist_cm < TRIGGER_DIST_CM) {
      set_period(PERIOD * 8 / 10); // open gate
      gate_open = true;
```

```
      TIMER0_CTL_R |= TIMER_CTL_TAEN;  // enable timer 0A
      // printf("%f\r\n", dist_cm);  // Debug info
    }
  }
}

void init_portF() {
  SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R5;  // enable clock to port F
  while (!(SYSCTL_PRGPIO_R & SYSCTL_PRGPIO_R5));  // wait for clock enabled
  GPIO_PORTF_AFSEL_R |= PF2 | PF4;  // enable alternate function
  GPIO_PORTF_PCTL_R |= GPIO_PCTL_PF2_T1CCP0 | GPIO_PCTL_PF4_T2CCP0;  // timer PW
  GPIO_PORTF_DIR_R |= PF2 | PF4;  // set PF2 as output
  GPIO_PORTF_DEN_R |= (PF2 | PF4);  // enable pins
}

void init_timer0A(uint32_t rate) {
  SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R0;  // enable timer0 clock
  while (!(SYSCTL_PRTIMER_R & SYSCTL_PRTIMER_R0));  // wait for clock enabled
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN;  // disable timer0A
  TIMER0_CFG_R |= TIMER_CFG_32_BIT_TIMER;  // set to 32bit mode
  TIMER0_TAMR_R |= TIMER_TAMR_TAMR_1_SHOT;  // set one-shot timer mode
  TIMER0_TAMR_R |= TIMER_TAMR_TACDIR;  // set count up mode
  TIMER0_TAILR_R = rate;  // set rate
  TIMER0_IMR_R |= TIMER_IMR_TATOIM;  // enable Timer0A interrupts
  NVIC_EN0_R |= EN0_19;
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN;  // disable timer0A
  TIMER0_ICR_R |= TIMER_ICR_TATOCINT; // clear interrupt flag
}

void init_timer1A(uint32_t rate) {
  SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R1;  // enable timer1 clock
  while (!(SYSCTL_PRTIMER_R & SYSCTL_PRTIMER_R1));  // wait for clock enabled
  TIMER1_CTL_R &= ~TIMER_CTL_TAEN;  // disable timer1A
  TIMER1_CFG_R |= TIMER_CFG_16_BIT;  // set to 16bit mode
  TIMER1_TAMR_R &= ~TIMER_TAMR_TACMR;  // clear capture mode
  TIMER1_TAMR_R |= TIMER_TAMR_TAAMS | TIMER_TAMR_TAMR_PERIOD;  // set period tim
  TIMER1_CTL_R |= TIMER_CTL_TAPWML;  // invert voltage
  TIMER1_TAMATCHR_R = RATE_16MHZ_10microHZ;  // send pulse of 10 microseconds
  TIMER1_TAILR_R = rate;  // time out before next sensor read
  TIMER1_CTL_R |= TIMER_CTL_TAEVENT_NEG;  // interrupt on neg edge to detect pos
  TIMER1_TAMR_R |= TIMER_TAMR_TAPWMIE;  // enable PWM timer interrupt
  TIMER1_IMR_R |= TIMER_IMR_CAEIM;  // enable Timer1A interrupts
  NVIC_EN0_R |= EN0_21;
  TIMER1_CTL_R |= TIMER_CTL_TAEN;  // enable timer1A
}

void init_timer2A(uint32_t rate) {
  SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R2;  // enable timer2 clock
  while (!(SYSCTL_PRTIMER_R & SYSCTL_PRTIMER_R2));  // wait for clock enabled
  TIMER2_CTL_R &= ~TIMER_CTL_TAEN;  // disable timer2A
  TIMER2_CFG_R |= TIMER_CFG_16_BIT;  // set to 16bit mode
```

```
  TIMER2_TAMR_R |= TIMER_TAMR_TACMR | TIMER_TAMR_TAMR_CAP;  // set one-shot time
  TIMER2_TAMR_R |= TIMER_TAMR_TACDIR;  // set count up mode
  TIMER2_CTL_R |= TIMER_CTL_TAEVENT_BOTH;  // capture both edges
  TIMER2_TAILR_R = rate;  // time out before next sensor read
  TIMER2_IMR_R |= TIMER_IMR_CAEIM;  // enable Timer2A interrupts
  NVIC_EN0_R |= EN0_23;
  TIMER2_CTL_R &= ~TIMER_CTL_TAEN;  // disable timer2A
}

void set_period(uint16_t period) {
  PWM0_0_LOAD_R = (period - 1)/2;  // count from zero to this number and back to
  PWM0_0_CMPA_R = (period - 1)/4;  // set up comparator A
}

void init_pwm(uint16_t period){
  SYSCTL_RCGCPWM_R |= SYSCTL_RCGCPWM_R0;  // enable clock to PWM0
  while (!(SYSCTL_PRPWM_R & SYSCTL_PRPWM_R0));  // wait for clock
  SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1;  // enable clock to GPIOB
  while (!(SYSCTL_PRGPIO_R & SYSCTL_PRGPIO_R1));  // wait for clock
  GPIO_PORTB_AFSEL_R |= PB6;  // enable alt funct on PB6
  GPIO_PORTB_PCTL_R |= GPIO_PCTL_PB6_M0PWM0;     // configure PB6 as M0PWM0
  GPIO_PORTB_DIR_R |= PB6;  // set as output
  GPIO_PORTB_DEN_R |= PB6;  // enable pin PB6
  SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV;  // divide system clock for pwm
  SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M;  // clear PWMDIV field
  SYSCTL_RCC_R |= SYSCTL_RCC_PWMDIV_64;  // configure for /64 divider
  PWM0_0_CTL_R = 0x00000000;  // set to immediate update
  //PWM0, Generator A (PWM0/PB6) goes to 0 when count==reload and 1 when count==
  PWM0_0_GENA_R = (PWM_0_GENA_ACTLOAD_ZERO | PWM_0_GENA_ACTZERO_ONE);
  PWM0_0_LOAD_R = (period - 1)/2;  // count from zero to this number and back to
  PWM0_0_CMPA_R = (period - 1)/4;  // set up comparator A
  PWM0_0_CTL_R |= (PWM_0_CTL_MODE | PWM_0_CTL_ENABLE); // set up count up/ down
  PWM0_ENABLE_R |= PWM_ENABLE_PWM0EN;  // enable PWM0
}
```

6.2 lab5.h

```
// AUTHORS: Daniel Sullivan, Jeffrey Lee
// COURSE: CSE 474
// PROJECT: Lab 5
// TITLE: lab5.h

#ifndef LAB_5_H_
#define LAB_5_H_

// NVIC Registers
#define NVIC_EN0_R              (*((volatile uint32_t *)0xE000E100))

// NVIC Values
#define INT_TIMER0A             35          // 16/32-Bit Timer 0A
#define INT_TIMER1A             37          // 16/32-Bit Timer 1A
#define INT_TIMER2A             39          // 16/32-Bit Timer 2A
```

```
// SYSCTL Registers
#define SYSCTL_RCGCGPIO_R       (*((volatile uint32_t *)0x400FE608))
#define SYSCTL_PRGPIO_R         (*((volatile uint32_t *)0x400FEA08))
#define SYSCTL_RCGCTIMER_R      (*((volatile uint32_t *)0x400FE604))
#define SYSCTL_PRTIMER_R        (*((volatile uint32_t *)0x400FEA04))
#define SYSCTL_RCGCPWM_R        (*((volatile uint32_t *)0x400FE640))
#define SYSCTL_PRPWM_R          (*((volatile uint32_t *)0x400FEA40))
#define SYSCTL_RCC_R            (*((volatile uint32_t *)0x400FE060))

// SYSCTL Values
#define SYSCTL_RCGCGPIO_R5      0x00000020  // GPIO Port F Run Mode Clock
                                            // Gating Control
#define SYSCTL_RCGCGPIO_R1      0x00000002  // GPIO Port B Run Mode Clock
                                            // Gating Control
#define SYSCTL_PRGPIO_R5        0x00000020  // GPIO Port F Peripheral Ready
#define SYSCTL_PRGPIO_R1        0x00000002  // GPIO Port B Peripheral Ready
#define SYSCTL_RCGCTIMER_R2     0x00000004  // 16/32-Bit General-Purpose Timer
                                            // 2 Run Mode Clock Gating Control
#define SYSCTL_RCGCTIMER_R1     0x00000002  // 16/32-Bit General-Purpose Timer
                                            // 1 Run Mode Clock Gating Control
#define SYSCTL_RCGCTIMER_R0     0x00000001  // 16/32-Bit General-Purpose Timer
                                            // 0 Run Mode Clock Gating Control
#define SYSCTL_PRTIMER_R2       0x00000004  // 16/32-Bit General-Purpose Timer
                                            // 2 Peripheral Ready
#define SYSCTL_PRTIMER_R1       0x00000002  // 16/32-Bit General-Purpose Timer
                                            // 1 Peripheral Ready
#define SYSCTL_PRTIMER_R0       0x00000001  // 16/32-Bit General-Purpose Timer
                                            // 0 Peripheral Ready
#define SYSCTL_RCGCPWM_R0       0x00000001  // PWM Module 0 Run Mode Clock
                                            // Gating Control
#define SYSCTL_PRPWM_R0         0x00000001  // PWM Module 0 Peripheral Ready
#define SYSCTL_RCC_USEPWMDIV    0x00100000  // Enable PWM Clock Divisor
#define SYSCTL_RCC_PWMDIV_M     0x000E0000  // PWM Unit Clock Divisor
#define SYSCTL_RCC_PWMDIV_64    0x000A0000  // PWM clock /64

// GPTM Registers
#define TIMER0_ICR_R            (*((volatile uint32_t *)0x40030024))
#define TIMER0_CTL_R            (*((volatile uint32_t *)0x4003000C))
#define TIMER0_CFG_R            (*((volatile uint32_t *)0x40030000))
#define TIMER0_TAMR_R           (*((volatile uint32_t *)0x40030004))
#define TIMER0_TAILR_R          (*((volatile uint32_t *)0x40030028))
#define TIMER0_IMR_R            (*((volatile uint32_t *)0x40030018))
#define TIMER0_ICR_R            (*((volatile uint32_t *)0x40030024))

#define TIMER1_CTL_R            (*((volatile uint32_t *)0x4003100C))
#define TIMER1_CFG_R            (*((volatile uint32_t *)0x40031000))
#define TIMER1_TAMR_R           (*((volatile uint32_t *)0x40031004))
#define TIMER1_TAMATCHR_R       (*((volatile uint32_t *)0x40031030))
#define TIMER1_TAILR_R          (*((volatile uint32_t *)0x40031028))
#define TIMER1_IMR_R            (*((volatile uint32_t *)0x40031018))
```

```
#define TIMER1_ICR_R            (*((volatile uint32_t *)0x40031024))

#define TIMER2_CTL_R            (*((volatile uint32_t *)0x4003200C))
#define TIMER2_ICR_R            (*((volatile uint32_t *)0x40032024))
#define TIMER2_TAR_R            (*((volatile uint32_t *)0x40032048))
#define TIMER2_CFG_R            (*((volatile uint32_t *)0x40032000))
#define TIMER2_TAMR_R           (*((volatile uint32_t *)0x40032004))
#define TIMER2_TAILR_R          (*((volatile uint32_t *)0x40032028))
#define TIMER2_IMR_R            (*((volatile uint32_t *)0x40032018))

// GPTM Values
#define TIMER_ICR_TATOCINT      0x00000001  // GPTM Timer A Time-Out Raw
                                            // Interrupt
#define TIMER_ICR_CAECINT       0x00000004  // GPTM Timer A Capture Mode Event
                                            // Interrupt Clear
#define TIMER_CTL_TAEN          0x00000001  // GPTM Timer A Enable
#define TIMER_CTL_TAPWML        0x00000040  // GPTM Timer A PWM Output Level
#define TIMER_CTL_TAEVENT_NEG   0x00000004  // Negative edge
#define TIMER_CTL_TAEVENT_BOTH  0x0000000C  // Both edges
#define TIMER_CFG_32_BIT_TIMER  0x00000000  // For a 16/32-bit timer, this
                                            // value selects the 32-bit timer
                                            // configuration
#define TIMER_CFG_16_BIT        0x00000004  // For a 16/32-bit timer, this
                                            // value selects the 16-bit timer
                                            // configuration
#define TIMER_TAMR_TAMR_1_SHOT  0x00000001  // One-Shot Timer mode
#define TIMER_TAMR_TACDIR       0x00000010  // GPTM Timer A Count Direction
#define TIMER_TAMR_TACMR        0x00000004  // GPTM Timer A Capture Mode
#define TIMER_TAMR_TAAMS        0x00000008  // GPTM Timer A Alternate Mode
                                            // Select
#define TIMER_TAMR_TAMR_PERIOD  0x00000002  // Periodic Timer mode
#define TIMER_TAMR_TAPWMIE      0x00000200  // GPTM Timer A PWM Interrupt
                                            // Enable
#define TIMER_TAMR_TAMR_CAP     0x00000003  // Capture mode
#define TIMER_IMR_CAEIM         0x00000004  // GPTM Timer A Capture Mode Event
                                            // Interrupt Mask
#define TIMER_IMR_TATOIM        0x00000001  // GPTM Timer A Time-Out Interrupt
                                            // Mask

// GPIO Registers
#define GPIO_PORTF_AFSEL_R      (*((volatile uint32_t *)0x40025420))
#define GPIO_PORTF_PCTL_R       (*((volatile uint32_t *)0x4002552C))
#define GPIO_PORTF_DIR_R        (*((volatile uint32_t *)0x40025400))
#define GPIO_PORTF_DEN_R        (*((volatile uint32_t *)0x4002551C))
#define GPIO_PORTB_AFSEL_R      (*((volatile uint32_t *)0x40005420))
#define GPIO_PORTB_PCTL_R       (*((volatile uint32_t *)0x4000552C))
#define GPIO_PORTB_DIR_R        (*((volatile uint32_t *)0x40005400))
#define GPIO_PORTB_DEN_R        (*((volatile uint32_t *)0x4000551C))

// GPIO Values
#define GPIO_PCTL_PF2_T1CCP0    0x00000700  // T1CCP0 on PF2
```

```
#define GPIO_PCTL_PF4_T2CCP0    0x00070000  // T2CCP0 on PF4
#define GPIO_PCTL_PB6_M0PWM0    0x04000000  // M0PWM0 on PB6

// PWM Registers
#define PWM0_0_CTL_R            (*((volatile uint32_t *)0x40028040))
#define PWM0_0_GENA_R           (*((volatile uint32_t *)0x40028060))
#define PWM0_0_LOAD_R           (*((volatile uint32_t *)0x40028050))
#define PWM0_0_CMPA_R           (*((volatile uint32_t *)0x40028058))
#define PWM0_ENABLE_R           (*((volatile uint32_t *)0x40028008))
#define PWM_ENABLE_PWM0EN       0x00000001  // MnPWM0 Output Enable

// PWM Values
#define PWM_0_GENA_ACTLOAD_ZERO 0x00000008  // Drive pwmA Low
#define PWM_0_GENA_ACTZERO_ONE  0x00000003  // Drive pwmA High
#define PWM_0_CTL_MODE          0x00000002  // Counter Mode
#define PWM_0_CTL_ENABLE        0x00000001  // PWM Block Enable

#endif  // LAB_5_H_
```

6.3 cstartup_M.c

```
/**************************************************
 *
 * This file contains an interrupt vector for Cortex-M written in C.
 * The actual interrupt functions must be provided by the application developer.
 *
 * Copyright 2007-2017 IAR Systems AB.
 *
 * $Revision: 112610 $
 *
 * Adapted by Daniel Sullivan, 2019
 * Lab5/cstartup_M.c
 *
 **************************************************/

#pragma language=extended
#pragma segment="CSTACK"

extern void __iar_program_start( void );

extern void NMI_Handler( void );
extern void HardFault_Handler( void );
extern void MemManage_Handler( void );
extern void BusFault_Handler( void );
extern void UsageFault_Handler( void );
extern void SVC_Handler( void );
extern void DebugMon_Handler( void );
extern void PendSV_Handler( void );
extern void SysTick_Handler( void );
extern void Timer0A_Handler( void );
extern void Timer0B_Handler( void );
extern void Timer1A_Handler( void );
```

```
extern void Timer2A_Handler( void );
extern void PortF_Handler( void );
extern void PortA_Handler( void );
extern void PortD_Handler( void );
extern void PortE_Handler( void );
extern void ADC0_Handler( void );

typedef void( *intfunc )( void );
typedef union { intfunc __fun; void * __ptr; } intvec_elem;

// The vector table is normally located at address 0.
// When debugging in RAM, it can be located in RAM, aligned to at least 2^6.
// If you need to define interrupt service routines,
// make a copy of this file and include it in your project.
// The name "__vector_table" has special meaning for C-SPY, which
// is where to find the SP start value.
// If vector table is not located at address 0, the user has to initialize
// the  NVIC vector table register (VTOR) before using interrupts.


#pragma location = ".intvec"
const intvec_elem __vector_table[] =
{
  { .__ptr = __sfe( "CSTACK" ) },
  __iar_program_start,

  NMI_Handler,
  HardFault_Handler,
  MemManage_Handler,
  BusFault_Handler,
  UsageFault_Handler,
  0,
  0,
  0,
  0,
  SVC_Handler,
  DebugMon_Handler,
  0,
  PendSV_Handler,
  SysTick_Handler,
  PortA_Handler,
  0,
  0,
  PortD_Handler,
  PortE_Handler,
  0,
  0,
  0,
  0,
  0,
  0,
```

```
  0,
  0,
  0,
  0,
  0,
  0,
  ADC0_Handler,
  0,
  Timer0A_Handler,
  Timer0B_Handler,
  Timer1A_Handler,
  0,
  Timer2A_Handler,
  0,
  0,
  0,
  0,
  0,
  0,
  PortF_Handler
};

#pragma call_graph_root = "interrupt"
__weak void NMI_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void HardFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void MemManage_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void BusFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void UsageFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SVC_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void DebugMon_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PendSV_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SysTick_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void Timer0A_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void Timer0B_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void Timer1A_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void Timer2A_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PortF_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
```

```
__weak void PortA_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PortD_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PortE_Handler ( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void ADC0_Handler ( void ) { while (1) {} }

void __cmain( void );
__weak void __iar_init_core( void );
__weak void __iar_init_vfp( void );

#pragma required=__vector_table
void __iar_program_start( void )
{
  __iar_init_core();
  __iar_init_vfp();
  __cmain();
}
```

References

[1] Elec Freaks. Ultrasonic ranging module hc-sr04. https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf.

[2] Texas Instruments. Tiva tm4c123gh6pm microcontroller data sheet. http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf, June 2014.

[3] OmniVision. Serial camera control bus functional specification. http://www4.cs.umanitoba.ca/~jacky/Teaching/Courses/74.795-LocalVision/ReadingList/ov-sccb.pdff, March 2002.

[4] OmniVision. Ov7670/ov7171 cmos vga (640x480) camerachip sensor with omnipixel technology. http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf, August 2006.

[5] Xnor.ai. Xnor model bundles. https://docs.ai2go.xnor.ai/model_bundles.html.